System design: Twitter

Suresh Podeti

9 min read · Oct 24, 2023

Photo by Alexander Shatov on Unsplash

Introduction

Twitter is a free **microblogging** social network where registered users post messages called "Tweets". Users can also like, reply to, and retweet public tweets.

Requirements

Functional

- **Post Tweets**: Registered users can post one or more Tweets on Twitter.
- **Delete Tweets**: Registered users can delete one or more of their Tweets on Twitter.
- **Like or dislike Tweets**: Registered users can like and dislike public and their own Tweets on Twitter.
- **Reply to Tweets**: Registered users can reply to the public Tweets on Twitter.
- **Search Tweets**: Registered users can search Tweets by typing keywords, hashtags, or usernames in the search bar on Twitter.
- **View user or home timeline**: Registered users can view the user's timeline, which contains their own Tweets. They can also view the home's timeline, which contains followers' Tweets on Twitter.
- **Follow or unfollow the account**: Registered users can follow or unfollow other users on Twitter.
- **Retweet a Tweet**: Registered users can Retweet public Tweets of other users on Twitter.

Non-functional

- **Availability**: Many people and organizations use Twitter to communicate time-sensitive information (service outage messages). Therefore, our service must be highly available and have a good uptime percentage.
- **Latency**: The latency to show the most recent top Tweets in the home timeline could be a little high. However, near real-time services, such as Tweet distribution to followers, must have low latency.
- **Scalability**: The workload on Twitter is read-heavy, where we have many readers and relatively few writers, which eventually requires the scalability of computational resources. We need high storage capacity to store and deliver Tweets posted by public figures to their millions of followers.
- **Reliability**: All Tweets remain on Twitter. This means that Twitter never deletes its content. So there should be a promising strategy to prevent the loss or damage of the uploaded content.

- **Consistency:** An effective technique is needed to offer rapid feedback to the user (who liked someone's post), then to other specified users in the same region, and finally to all worldwide users linked to the Tweet.

Design

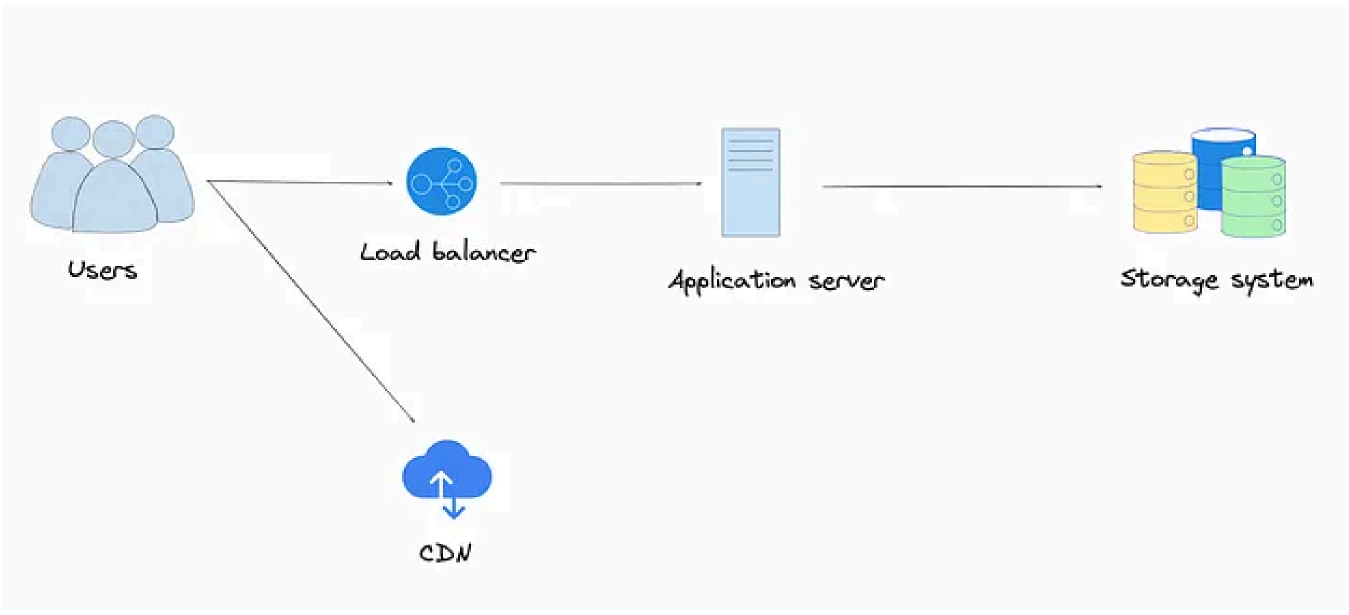

Fig 1.0: High level design of twitter

- **Users** post Tweets delivered to the server through the load balancer. Then, the system stores it in persistent storage.
- **CDN** is situated near the users to provide requested data with low latency. When users search for a specified term or tag, the system first searches in the CDN proxy servers containing the most frequently requested content.
- **Load balancer** chooses the operational application server based on traffic load on the available servers and the user requests.
- **Storage system** represents the various types of storage (SQL-based and NoSQL-based).
- **Application servers** provide various services and have business logic to orchestrate between different components to meet our functional requirements.

Storage systems

Google Cloud

Why is it used?

To better analyse and manage the data

What data is stored?

Logs (client events, Tweet events, and timeline events), MySQL and Manhattan backups, ad targeting and analytics, user engagement predictions, social graph analysis, and so on..

Data is stored in the **BigQuery** (Google cloud service), a fully managed and highly scalable server-less data warehouse. Twitter uses the **Presto** (distributed SQL query engine) to access data from Google Cloud (BigQuery, Ad-hoc clusters, Google cloud storage, and so on).

Learn more about how twitter uses bigQuery:

Twitter modernized its approach to data processing using Google Cloud services like Big Query. |...

Twitter transformed its approach to data processing using Google Cloud services. In this blog, Pradip Thachile, cloud...

cloud.google.com

Manhattan

Twitter launched its own general-purpose real-time d**istributed key-value store**, called Manhattan.

What data is stored?

Tweets, Twitter accounts, direct messages, and so on

Manhattan uses **RocksDB** as a storage engine responsible for storing and retrieving data within a particular node. RocksDB is a storage engine, not a standalone database. RocksDB is often used as the underlying storage engine for other database systems.

Learn more about twitter's manhattan database:

Manhattan, our real-time, multi-tenant distributed database for Twitter scale

Manhattan, our real-time, multi-tenant distributed database for Twitter scale

Manhattan, our real-time, multi-tenant distributed database for Twitter
scaleblog.twitter.com

Blobstore

What data is stored?

Twitter built the Blobstore storage system to store **photos** attached to Tweets. It also stores **videos, binary files, and other objects**.

After a specified period, the server checkpoints the in-memory data to the Blobstore as durable storage.

Learn more about twitter's blob storage:

Blobstore: Twitter's in-house photo storage system

Blobstore: Twitter's in-house photo storage system

Blobstore: Twitter's in-house photo storage systemblog.twitter.com

SQL-based databases

What databases?

MySQL and PostgreSQL

What data is stored?

Data where it needs strong consistency, ads exchange, and managing ads campaigns.

Twitter also uses **Vertica** to query commonly aggregated datasets and Tableau dashboards. Around 2012, Twitter also built the **Gizzard framework** on top of **MySQL for sharding**, which is done by partitioning and replication.

Learn more about how twitter uses Gizzard for MySQL sharding:

Introducing Gizzard, a framework for creating distributed datastores

Introducing Gizzard, a framework for creating distributed datastores

Introducing Gizzard, a framework for creating distributed datastoresblog.twitter.com

Kafka and Cloud dataflow

Twitter evaluates around 400 billion real-time events and generates petabytes of data every day. For this, it processes events using Kafka on-premise and uses **Google Dataflow jobs** to handle **deduping** and real-time aggregation on Google Cloud. After aggregation, the results are stored for ad-hoc analysis to BigQuery (data warehouse) and the serving system to the Bigtable (NoSQL database). Twitter **converts Kafka topics into Cloud Pub-sub topics using an event processor**, which helps avoid data loss and provides more scalability.

FlockDB

Twitter uses FlockDB, a graph database.

What data is stored?

Relationships like User's followers, who the user follows, whose notifications the user has to receive, and so on. Twitter stores this relationship in the form of a graph.

Learn more about twitter's FlockDB:

Introducing FlockDB

Introducing FlockDB

Introducing FlockDBblog.twitter.com

Apache Lucene

Twitter constructed a **search service** that indexes about a trillion records and responds to requests within 100 milliseconds. Twitter stores a real-time index (recent Tweets during the past week) in RAM for low latency and quick updates. The full index is a hundred times larger than the real-time index. However, Twitter performs batch processing for the full indexes.

Cache

Twitter has been used as multi-tenant (multiple instances of an application have the shared environment) Twitter Memcached (Twemcache) and Redis (Nighthawk) clusters for caching. Due to some issues such as unexpected performance, debugging difficulties, and other operational hassles in the existing cache system (Twemcache and Nighthawk), Twitter has started to use the **Pelikan** cache. This cache gives high-throughput and low latency.

Pelikan introduced **Segcache**, a highly scalable and memory-efficient back-end server for small objects in large-scale caches, where the average object size is 200 to 300 bytes. Unlike other solutions like Memcache and Redis, which have 56 bytes of metadata per object, **Pelikan reduced the metadata size to just 38 bytes, optimizing memory usage**.

Observability

Monitoring and Alerting

Twitter uses various tools to monitor services' health, such as providing alerts and support for multiple issues. Twitter's Alert system notifies broken or degraded services triggered by the set metrics.

Twitter used the LonLens service that delivers visualization and analytics of service logs. Later, it was replaced by Splunk Enterprise, a central logging system.

Tracing

Tracing billions of requests is challenging in large-scale real-time applications. Twitter uses **Zipkin**, a distributed tracing system, to trace each request (spent time and request count) for multiple services. Zipkin selects a portion of all the requests and attaches a lightweight trace identifier. This sampling also reduces the tracing overhead.

Twitter uses Zookeeper to store service registry, Manhattan clusters' topology information, metadata, and so on. Twitter also uses it for the leader election on the various systems.

Heavy hitter problem

Twitter hosts millions of accounts, some with millions of followers. When these accounts post Tweets, millions engage quickly. Handling billions of interactions, like views and likes, poses the heavy hitter problem.

A single counter for each specific operation on the particular Tweet is not enough. It's challenging to handle millions of increments or decrements requests against a particular Tweet in a single counter. Therefore, we need multiple **distributed**

counters to manage burst write requests (increments or decrements) against various interactions on celebrities' Tweets.

Each counter has several shards working on different computational units. These distributed counters are known as **sharded counters**. These counters also help in another real-time problem named the **Top-k** problem.

Complete design

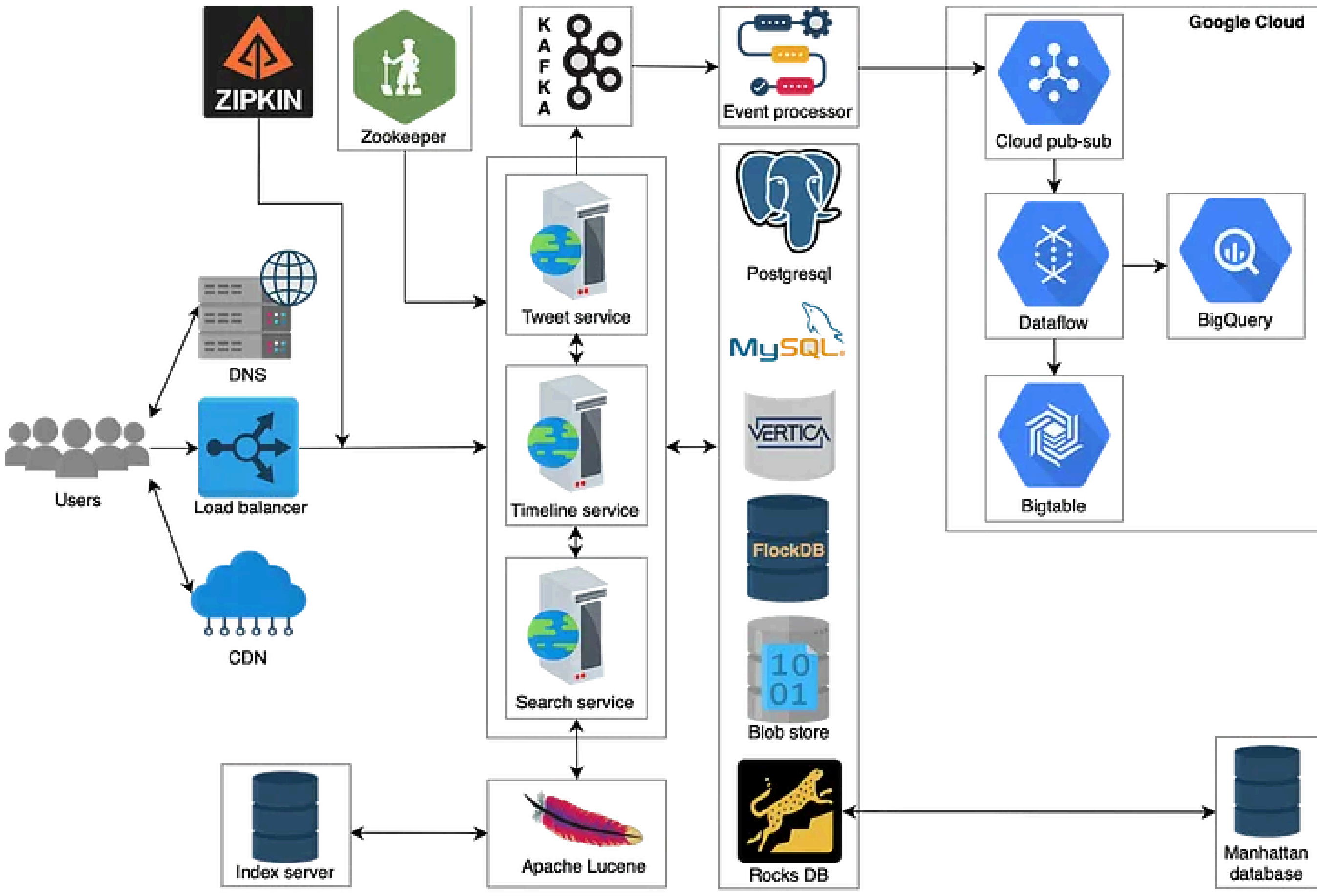

Fig 2.0: Twitter design overview

- First, end users get the address of the nearest load balancer from the local DNS.
- Load balancer routes end users' requests to the appropriate servers according to the requested services. Here, we'll discuss the Tweet, timeline, and search services.
- **Tweet service:** When end users perform any operation, such as posting a Tweet or liking other Tweets, the load balancers forward these requests to the server handling the Tweet service. Consider an example where users post Tweets on Twitter using the `/postTweet` API. The server (Tweet service) receives the requests and performs multiple operations. It identifies the attachments (image,

video) in the Tweet and stores them in the Blobstore. Text in the Tweets, user information, and all metadata are stored in the different databases (Manhattan, MySQL, Postgresql, Vertica). Meanwhile, real-time processing, such as pulling Tweets, user interactions data, and many other metrics from the real-time streams and client logs, is achieved in the Apache Kafka.

- Later, the data is moved to the cloud pub-sub through an event processor. Next, data is transferred for deduping and aggregation to the BigQuery through Cloud Dataflow. Finally, data is stored in the Google Cloud Bigtable, which is fully managed, easily scalable, and sorted keys.
- **Timeline service**: Assume the user sends a home timeline request using the `/viewHome_timeline` API. In this case, the request is forwarded to the nearest CDN containing static data. If the requested data is not found, it's sent to the server providing timeline services. This service fetches data from different databases or stores and returns the Top-k Tweets. This service collects various interactions counts of Tweets from different sharded counters to decide the Top-k Tweets. In a similar way, we will obtain the Top-k trends attached in the response to the timeline request.
- **Search service**: When users type any keyword(s) in the search bar on Twitter, the search request is forwarded to the respective server using the `/searchTweet` API. It first looks into the RAM in Apache Lucene to get real-time Tweets (Tweets that have been published recently). Then, this server looks up in the index server and finds all Tweets that contain the requested keyword(s). Next, it considers multiple factors, such as time, or location, to rank the discovered Tweets. In the end, it returns the top Tweets.
- We can use the Zipkin tracing system that performs sampling on requests. Moreover, we can use Zookeeper to maintain different data, including configuration information, distributed synchronization, naming registry, and so on.

Client-side Load Balancer for Twitter

We discussed Twitter's design with a dedicated load balancer in the previous section. While effective in some cases, it may not be ideal for Twitter's diverse and large-scale services. This is because Twitter offers a variety of services on a large scale, using numerous instances and **dedicated load-balancers are not a suitable choice for such systems**.

Twitter's **initial monolithic design** with **Ruby on Rails and a MySQL database** became problematic as it scaled. The drawbacks included challenges in updating services, potential for one upgrade to break another, rising hardware costs, and complex failure recovery. To address these issues, Twitter transitioned to a **microservices architecture** to enhance scalability and flexibility.

Client-side load balancing eliminates the need for an intermediary load-balancing system. Instead, **each requesting node or client has its built-in load balancer to choose a suitable service instance.** This approach enhances flexibility and scalability, with services registering themselves in a service registry for visibility to other services.

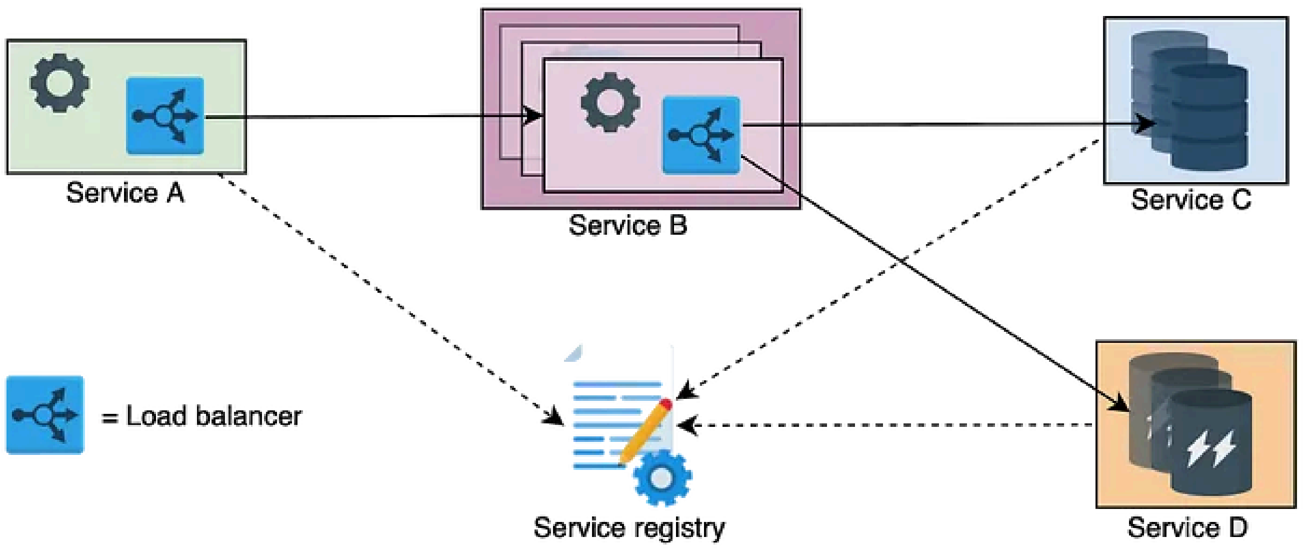

Fig 3.0: How client-side load balancing works

In this client-side load balancing approach, every service makes its own load-balancing decisions, reducing the need for central infrastructure. Service A selects the best instance of Service B using its built-in load balancer. This method minimizes hardware, lowers network latency, and enhances reliability. Popular services like Twitter, Yelp, and Netflix use client-side load balancing, providing a better quality of experience with fewer bottlenecks.

Learn more about twitter's client side load balancing:

Deterministic Aperture: A distributed, load balancing algorithm

As parts of the Twitter application grow, we can scale demands on capacity by adding more instances or replicas to a...

blog.twitter.com

System Design Interview · Software Architecture · Software Architect · Twitter · Distributed Systems

Edit profile

Written by Suresh Podeti

1.2K Followers

Aspiring Software Architect | SDE III (Golang) @JungleeGames| Ex-Byju's | Co-founder @Rupant Tech. | Ex-Synopsys | I.I.T Kharagpur | 7+ Years of Experience

Recommended from Medium